ブロック-テキスト型による段階的なプログラミング学習の提案

池見 樹[*1]・工藤 隼佑[*1]・森本 悠介[*1]・若松 大樹[*1]
指導教員：渡邉 寿昭[*2]
Email: Toshiaki_Watanabe@education.metro.tokyo.jp

*1：東京都立多摩科学技術高等学校 3 年
*2：東京都立多摩科学技術高等学校

◎Key Words　　プログラミング学習，ビジュアル型プログラミング，段階的学習

1. はじめに

現在，情報技術の発展に伴い，家電製品や自動車をはじめとする様々な製品にコンピュータが組み込まれており，人々の生活の利便性と豊かさの向上に大きく寄与している。これらのコンピュータを制御するためにはプログラムに関する知識が不可欠であり，プログラミングは情報社会を支える基盤技術の一つとなっている。近年では，生成 AI の普及により，AI を利用したコーディングも行われるようになったが，AI に依存することによって，十分なプログラミング能力の向上につながらない可能性が指摘されている(2)。したがって，このような環境においても，より実践的なプログラミング能力を身に付け，プログラムの理解を深めることは依然として重要である。そのため，初学者が無理なく知識・技能を習得できる学習環境の整備が求められている(1)。

プログラミング教育の手法として，テキスト型プログラミングを用いるものと，ビジュアル型プログラミングを用いるものがある。ビジュアル型プログラミングは視覚的に理解しやすいブロックやアイコンを組み合わせてプログラムを構成する方法である。この手法では，文法エラーを意識せずに試行錯誤できる点や，プログラムの構造理解を促しやすい点から，初学者の導入教材として有効であることが報告されている(3)。

一方，テキスト型プログラミングは自由度が高く，実際のソフトウェア開発でも広く利用されていることから，実用的なスキルの獲得に不可欠である。しかし，記述ミスの発生やエラー原因の特定・理解に時間を要し，学習者が達成感を得にくいという問題がある。また，コード量が多くなるため，初学者が学習初期に用いるには負担が大きい。

これらの特徴から，初学者がプログラミングを学習する際には，ビジュアル型プログラミングからテキスト型プログラミングへ段階的に移行する学習方法の有効性が示されている(4)。

また，近年では，ビジュアル型とテキスト型を同時に扱う学習環境の開発も進められており，その一つに Google が提供する Blockly がある。この環境は，ブロックで作成したプログラムを JavaScript や Python のコードへ変換する機能を備えている。しかし，このような環境は学習者の理解度に応じてテキスト型へ移行させる支援機能が十分でなく，変換されたコードの理解を促す学習支援機能も不足しているとの指摘がある(4)。

2. 研究目的

本研究では，初学者がビジュアル型プログラミングからテキスト型プログラミングへ段階的に移行できる学習環境を開発することを目的とする。また，学習者が自律的にプログラムへの理解を深め，継続的な学習を行える環境の実現を目指す。

3. システムの設計

本研究では，利便性の観点から，VS Code の拡張機能として学習環境を実装する。

3.1 全体の構成

本研究で開発した VS Code の拡張機能は，エディタ内に「ブロック型エディタ」と「テキスト型エディタ（Python）」を並列配置し，同一のプログラムを複数表現で同時に扱える学習環境を提供する。内部構造として，ブロック表現とテキスト表現の間に中間表現（Intermediate Representation：IR）を配置する。ブロック表現からテキスト表現への変換は，「ブロック表現→IR→Python」の変換パイプラインによって実現している。一方，テキスト表現からブロック表現への変換については，Python ソースコードを抽象構文木（Abstract Syntax Tree：AST）へ変換した後，IR を経由してブロック表現へ変換する「Python→AST→IR→ブロック表現」のパイプラインを設計した。これにより，片方の表現を編集すると IR を介してもう一方へ意味的に整合性のとれた変換が行われる。

UI は，左ペインに Blockly によるブロック型エディタ，右ペインに Python エディタを配置する構成とする。また，1 つの IR から複数言語のテキストコードを生成・同期する拡張も想定する。

3.2 ブロック型→テキスト型(Python)変換機能

ブロック型エディタには，「出力（print）」「変数の代入」「条件分岐（if 文）」「繰り返し（while，for）」「算術演算」など，初学者向けに限定したブロックを用意する。各ブロックは，ブロック固有 ID（blockID），ブロック種別（print，assign，if，while，など），引数スロットに設定された文字列や式，および接続構造（前後関係・入れ子関係）を内部に保持する。

ブロックを IR に変換し，IR から生成された Python コードは右側のテキストエディタに即座に反映される。学習者がブロックを追加・削除・編集するたびに IR が更新され，それに基づき Python コードも再生成されるため，「ブロックで組んだ処理がどのようなテキストコードに対応するのか」をリアルタイムに確認できる。

3.3 テキスト型 → ブロック型変換機能

本拡張機能の設計では，Python コードからブロック構成を再構成する機能も組み込む。想定する手順は次のとおりである。学習者がテキストエディタ上で Python コードを編集し，構文解析器により抽象構文木（AST）を構築する。AST を IR へ変換し（例：ast。If ノードを IR の If ノードへ写像），言語依存の構造を言語非依存の IR へ置き換える。その後，IR から Blockly 上のブロック構成を再生成する。IR には可能な限り元の blockID や行番号などのメタデータを保持し，既存ブロックとの対応づけを行う。対応ブロックが存在する場合は更新し，存在しない場合は新規生成してワークスペースへ配置する。

一方，構文的には正しいが対応するブロック定義が存在しない高機能な構文（例：一部標準ライブラリの呼び出し等）については，ブロックへの変換を行わず，ブロックエディタ上部に警告メッセージを表示して学習者に通知する設計とする。

3.4 IR を中心とした双方向同期設計

ブロックとテキストの相互変換はすべて IR を経由して行う。この設計により，Python に加えて Java などの多言語を導入する場合にも，IR の設計を大きく変えずに多言語対応が可能となる。さらに，IR ノードに「どのブロックから生成されたか」「Python コードの何行目に対応するか」といったメタデータを付与することで，将来的にブロックとコード行の相互ハイライト等の視覚的マッピング機能を実装しやすくなる。

3.5 レベル別 UI による段階的学習支援

本拡張機能は，学習者の習熟度に応じてブロック表示を切り替える「レベル別 UI」を備える。Webview 上部のレベルボタン操作により，ツールボックス内のブロックに加えて，ワークスペース上のブロックラベルも切り替える。

・レベル 1：日本語中心の表現
（例：「出力する」）
・レベル 2：日本語＋コードの併記
（例：「出力（print）する」）
・レベル 3：コード中心の表現
（例：print）

図 3-1 UI（レベル 1）

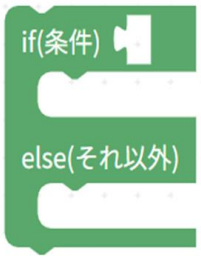

図 3-2 UI（レベル 2）

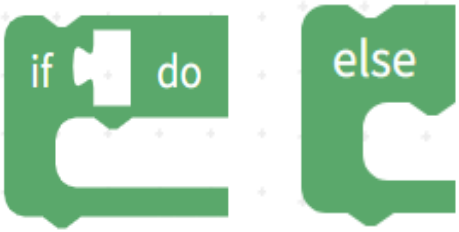

図 3-3 UI（レベル 3）

内部の IR はレベルに依存せず同一であるため，見た目のみを切り替えながら意味的には同じプログラムとして扱える。この設計により，「自然言語に近い見た目 → 混合表現 → 純粋なコード表現」という足場かけを通じて，初学者が無理なくテキスト型プログラミングに移行することを支援する。

4. 評価実験

本研究の実証実験は，2026 年 2 月 6 日～12 日の間に，実験を実施した。参加者はプログラミング経験のない 9 名（小学生 1 名，中学生 3 名，高校生 4 名，大学生 1 名）である。

4.1 実験方法

参加者を，本制作物を用いて学習する群（パターン 1）と，従来どおりテキスト（コード）ベースで学習する群（パターン 2）の 2 グループに分け，同程度の難易度の課題を取り組ませた。学習効果を検証するため，学習の前後に内容が酷似したテスト（事前テスト／事後テスト）を実施し，得点の変化を比較した。

4.2 実験環境

両条件間で使用する指導書の基本構成及び，課題内容を統一した。ただし，パターン 1 では本制作物の利用方法の理解をするため，ブロック操作に関する簡易的な説明を追加した。また，実施するテストの問題も両条件で統一したが，プログラムの説明方法のみを変更した。なお，参加者の条件振り分けは参加者本人の希望を考慮し決定している。

使用した指導書を作成するにあたって，多摩科学技術高等学校の IT 領域指導書を一部参考にした。

図 4-1 パターン 1 の指導書の例

図 4-2 パターン 2 の指導書の例

4.3 評価方法

事前テストおよび事後テストの得点を用い，得点差（事後テスト得点 − 事前テスト得点）を学習効果の指標とした。事前テスト，事後テストの構成はパターン 1，パターン 2 で同一である。テストは全 6 問からなり，プログラムに関する知識を問う設問と，フローチャート作成を通じてプログラミング的思考の理解を問う設問を同程度の割合で含めた。事前テストは選択式や並び替え問題を中心に初学者でも取り組みやすい難易度とし，事後テストは記述式を中心にして難易度を上げた。問題は、参考文献[5]をもとに作成した。

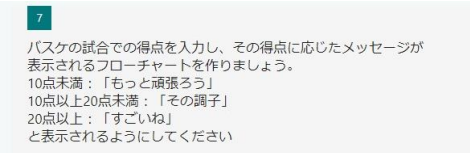

図 5-1 事前問題の例(フローチャート)

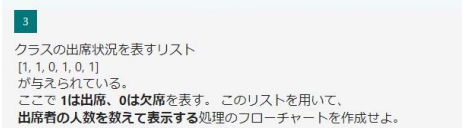

図 5-2 事後問題の例(フローチャート)

5. 結果

本制作物を用いて学習を行ったパターン 1（参加者 5 名：BA, BB, BC, BD, BE）と，従来どおりテキスト（コード）ベースで学習を行ったパターン 2（参加者 4 名：CA, CB, CC, CD）の結果を比較したところ、図 6 のようになった。

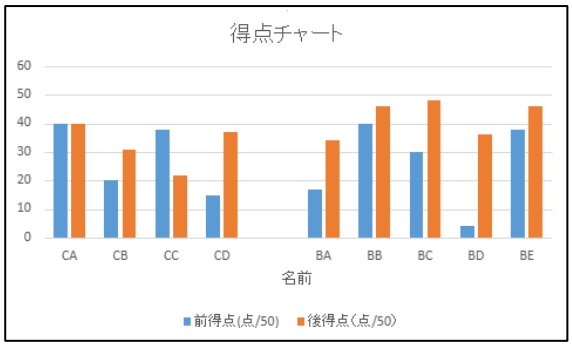

図 6 得点変化量

パターン 1 とパターン 2 を比較すると，パターン 1 では平均 16.2 点，パターン 2 では平均 4.25 点の上昇が見られた。よって，パターン 1 の得点向上幅は，パターン 2 に比べて大きい傾向が見られた。

6. 考察

本研究は，中高生を中心としたプログラミング未経験者を対象に，ブロック表現とテキスト表現の相互翻訳機能を備えた学習用エディタを開発することを目的とした。実証実験の結果，本制作物の使用者（パターン 1）の得点向上幅は，従来どおりテキスト（コード）ベースで学習した参加者（パターン 2）より大きい傾向が見られた。この要因として，複数段階のブロック表現により，学習者が理解度に応じて表現の抽象度を調整しながら学習できた可能性がある。また，ブロックとテキストの対応関係を即時に確認できることで，構文と意味の結び付きが促進され，テキスト型への移行に伴う負荷が軽減された可能性がある。以上より，本制作物は短期間の学習において，従来法と比較して学習効率を高める可能性が示唆された。

また、本制作物を使用した学習者では得点の低下が見られなかった一方で，従来法では得点が低下した参加者が見られた。このため，条件間の平均値の差は，従来法における得点の低下によって生じた可能性もある。今後は，平均値に加えて個人ごとの変化量の分布を提示し，ばらつきも含めて評価していく。

7. 今後の課題

本結果は短期的な傾向ではあるが，段階的表現と即時翻訳が学習負荷を下げ，誤りの自己修正を促した可能性がある。一方で本実験は参加者が計 9 名と少ないかつ 2 パターン間で年齢などが統一されていない，また参加者の希望に基づいて条件を割り当てたため（無作為割り付けではない），条件間の差が介入効果のみを反映していない可能性がある。さらに，事前・事後テストは内容が近いものの同一ではないため，テスト難易度差の影響を受ける可能性がある。今後の課題として，以下のことが必要である。

・評価の妥当性を高めるため，参加者数（サンプル数）を増やした上で，学習時間・使用機器等の条件を可能な範囲で統制した長期的な検証を行う。

・学習継続率，課題完遂率，理解度テストの再現性などの評価指標の整備を行うことで，時系列で評価できる枠

組みを構築する。

・参加者の年齢層・経験レベル・学習目的を拡大し、より広い学習者層に適用可能な環境へ発展させる。

・他の有効な学習方法（教材・支援機能）との比較実験を通じて，本機能の有用性および強みを明確化する。

・対応する実装プログラミング言語を増やし，学習者の目的や学習段階に応じた言語選択を可能にする。

・ブロックとテキストの対応関係の提示方法（行数表示）を改善し，関連性をより理解しやすくする。

・学習者の理解度を確認できる実力テスト機能を実装し，学習の定着度を継続的に測定できるようにする。また，それにより学習のモチベーション維持を図る。

8. おわりに

本研究では，ビジュアル型プログラミングからテキスト型プログラミングへ段階的に移行できる学習支援の開発を目的として，ブロック表現とテキスト表現の相互翻訳機能を備えた学習用エディタを開発した。実証実験の結果，本制作物の使用者は従来法と比較して得点向上幅が大きい傾向を示し，短期間の学習における有効性が示唆された。ただし，本成果は短期評価に基づくため，学習効果の持続性や理解度の定着度を確認する長期的な評価が不可欠である。また，評価対象の年齢層・経験レベル・学習目的を拡大し，教材設計や機能設計の最適化を進めることで，より広い学習者層に適用可能な環境へ発展させたい。

参考文献

(1) 文部科学省，「小学校プログラミング教育の趣旨と計画的な準備の必要性について」，2019.（参照日：2025 年 11 月 28 日）

(2) Prather, J., Reeves, B. N., Leinonen, J., MacNeil, S., Randrianasolo, A., Becker, B. A., Kimmel, B., Wright, J., & Briggs, B. (2024). The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers. Proceedings of the International Computing Education Research Conference (ICER 2024).

(3) David Weintrop and Uri Wilensky, “Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms,” ACM Transactions on Computing Education,18(1), Article 3, 2017.（参照日：2026年2月27日）

(4) Kölling, M., Brown, N. C. C., & Altadmri, A. (2015). Frame-Based Editing: Easing the Transition from Blocks to Text-Based Programming. Proceedings of the Workshop in Primary and Secondary Computing Education.

(5) 会津大学,「管理学研究科ニュース 2019 年 4 月」，2019.（参照日：2026 年 2 月 27 日）https://www.jc.u-aizu.ac.jp/news/management/gr/2019/04.pdf